Tervezési minták

A tervezési minták (design patterns) a szoftverfejlesztésben olyan általánosan elfogadott és ismert megoldások, amelyek ismétlődő problémákra adnak hatékony válaszokat. Ezek a minták nem konkrét kódot jelentenek, hanem inkább egyfajta iránymutatásokat adnak arra, hogyan oldhatunk meg gyakori problémákat egy szoftver rendszerben. A tervezési minták segítenek abban, hogy a kód tisztább, átláthatóbb, újrahasznosíthatóbb és karbantarthatóbb legyen.
A tervezési minták három fő kategóriába sorolhatók:

1. Creational (Kreációs) minták

Ezek a minták a példányosítás (objektumok létrehozása) problémáira adnak megoldást. A cél, hogy a rendszer ne legyen túl szoros kapcsolatban az objektumok konkrét típusaival, így könnyebben módosítható és bővíthető legyen. A leggyakoribb kreációs minták:

- **Singelton**
- **Factory Method**
- **Abstract Factory**
- **Builder**
- **Prototype**

2. Structural (Szerkezeti) minták

A szerkezeti minták a különböző osztályok és objektumok közötti kapcsolatokat és azok szerkezetét célozzák meg. Segítenek abban, hogy a rendszerek könnyebben bővíthetőek és módosíthatóak legyenek. A leggyakoribb szerkezeti minták:

- **Adapter**
- **Bridge**
- **Composite**
- **Decorator**
- **Facade**
- **Flyweight**
- **Proxy**

3. Behavioral (Viselkedési) minták

A viselkedési minták a szoftverkomponensek közötti kommunikációt és interakciót célozzák meg. Segítenek abban, hogy a program különböző részei hatékonyan és rugalmasan működjenek együtt. A leggyakoribb viselkedési minták:

- **Chainof responsibility**

- **Command**
- **Iterator**
- **Mediator**
- **Memento**
- **Observer**
- **State**
- **Strategy**
- **Template**
- **Visitor**

Ezek közül hármat szeretnék bemutatni.

Prototype

A „Prototype” (prototípus) tervezési minta célja, hogy lehetővé tegye egy objektum klónozását ahelyett, hogy új példányokat hoznánk létre a szokásos konstruktorokkal. Ez különösen hasznos lehet, ha egy objektum inicializálása költséges vagy bonyolult, és nem szeretnénk új példányokat létrehozni minden alkalommal, hanem inkább a már meglévő példányok másolatát használjuk. Az alapelv az, hogy egy „prototípus” objektumot hozunk létre, amelyről másolatokat készítünk, így az objektumok gyorsabban és kevesebb erőforrással másolhatók.

A minta akkor hasznos, ha az objektumok létrehozása bonyolult vagy költséges, illetve ha az objektumok állapotát dinamikusan kell változtatni anélkül, hogy új példányokat hoznánk létre. A Prototype minta gyakran alkalmazható olyan alkalmazásokban, ahol az objektumok hasonlóak, de apróbb különbségekkel rendelkeznek.

A tervezési minta általában a következő elemeket tartalmazza:

- **Prototype Interface**: A prototípusok klónozási képességét deklaráló interface.
- **Concrete Prototype**: Az objektumok, amelyeket klónozni lehet.
- **Client**: Az a komponens, amely a klónozást végzi és használja az objektumokat.

A minta előnye, hogy gyorsabban hozhatóak létre az objektumok, mivel nem szükséges újraépíteni őket teljesen. Hátránya, hogy a klónozott objektumoknak kompatibilisnek kell lenniük a prototípussal, ami néha bonyolulttá válhat, különösen akkor, ha az objektumok összetett állapotokkal rendelkeznek

Bridge

A „Bridge” tervezési minta egy strukturális minta, amely segít elválasztani egy absztrakciót a hozzá kapcsolódó implementációtól, lehetővé téve, hogy azok külön-külön változtathatóak legyenek. A Bridge minta célja, hogy elkerülje az öröklődés bonyolult hierarchiáját, különösen akkor, mikor többféle implementációs változat is létezik, de nem akarjuk az összes lehetséges kombinációt kódba égetni.

Mikor használjuk?
A Bridge minta akkor hasznos, ha az implementációk és azok absztrakciói szoros kapcsolatban állnak egymással, de a különböző verziók vagy implementációk könnyen módosíthatók, vagy bővíthetők anélkül, hogy az absztrakciót és az implementációt összekevernénk. Ha tehát különböző platformokon vagy típusokon kell dolgozni, és fontos, hogy az absztrakciók változtatása ne befolyásolja az implementációkat, vagy fordítva, akkor a Bridge minta ideális választás.

Előnye hogy lehetővé teszi az absztrakciók és az implementációk független fejlesztését, mivel nem szükséges többféle öröklődéssel foglalkoznunk, így csökkenti a komplexitást, és könnyen bővíthető ha új implementációk, vagy absztrakciók kerülnek hozzáadásra. Hátránya viszont hogy több osztály szükséges, amely bonyolíthatja a kódunkat.

Iterátorok

Az **Iterator** tervezési minta lehetővé teszi kollekciók (pl. listák, tömbök) elemeinek szekvenciális bejárását anélkül, hogy feltárnánk a belső implementációjukat. A minta fő elemei:

- **Iterator interfész**: Meghatározza a bejáráshoz szükséges műveleteket (pl. következő elem, van-e még elem).
- **Concrete Iterator**: Az interfész konkrét megvalósítása adott kollekcióhoz.
- **Aggregate interfész**: Kollekciók létrehozása és kezelésének módját definiálja.
- **Concrete Aggregate**: Az adott kollekció megvalósítása.

Előnyök: egységes hozzáférés különböző kollekciókhoz, kód újrafelhasználhatósága. Hátrányok: többletkomplexitás, ha egyszerű bejárás elegendő.

Memento

A **Memento** tervezési minta célja egy objektum állapotának mentése és későbbi visszaállítása anélkül, hogy a belső részleteit felfednénk. Ez a minta három fő szereplőt tartalmaz:

- **Originator**: Az objektum, amelynek az állapotát menteni és visszaállítani szeretnénk.
- **Memento**: A tárolt állapotot képviselő osztály.
- **Caretaker**: A mentett állapotok kezelője.

Előnyök: állapot mentése és visszaállítása egyszerűen. Hátrány: magas memóriaköltség komplex objektumok esetén.

Miért fontosak a tervezési minták?

A tervezési minták használata segít abban, hogy a programozók:

- **Hatékonyabban dolgozzanak**: A minták ismert és tesztelt megoldásokat kínálnak a leggyakoribb problémákra.
- **Rugalmasabb és bővíthetőbb rendszereket építsenek**: Az új funkciók hozzáadása vagy a meglévő funkciók módosítása a minták alkalmazásával könnyebbé válik.
- **Jobban olvasható kódot írjanak**: A tervezési minták segítenek a kód strukturálásában, ami megkönnyíti a más fejlesztők számára való megértést és karbantartást.

A tervezési minták tehát kulcsfontosságúak a komplex szoftverek fejlesztésében, és azok tudatos alkalmazása segíti a kód tisztán tartását, könnyebb karbantartását és jövőbeli bővítését.